

## Reverse Engineering Report – Crackme Challenge

Author: Ali Rashchi

Date: 2025-08-09

Challenge: <https://crackmes.one/crackme/682113c96297cca3ff7d7834>

Title: Lilsan44444's Validator (easy)

Platform: Windows x86-64

Difficulty: 1.0

Language: C/C++

### Objective

The objective of this challenge was to bypass both the debugger detection mechanism and the password validation check within the target executable, ultimately achieving the 'success' state without providing the correct base64-encoded password.

### Tools Used

- x64dbg
- Windows 11 CMD

### Methodology

- \*\*Initial Analysis\*\***: Loaded the crackme in x64dbg and located the anti-debugger function using the symbol `IsDebuggerPresent`.
- \*\*Debugger Bypass\*\***: Patched the instruction `setne al` with `xor eax, eax` to force the zero flag state and avoid debugger detection.
- \*\*Password Check Bypass\*\***: Identified the `test al, al` followed by a conditional jump (`je`) to the success message. Patched the conditional jump with NOP instructions to always execute the success branch.
- \*\*Final Test\*\***: Executed the patched binary to verify that it bypasses both checks and displays the success message.

### Screenshots and Addresses

Screenshot showing debugger check function before patch (Address: 0x00007FF6F3C71463)

00007FF6F3C71454	48:83EC 20	sub rsp,20	
00007FF6F3C71458	48:8B05 256E0000	mov rax,qword ptr ds:[<IsDebuggerPresent>]	rax:___init
00007FF6F3C7145F	FD00	call rax	rax:___init
00007FF6F3C71461	85C0	test eax,eax	
00007FF6F3C71463	0F95C0	setne al	
00007FF6F3C71466	48:83C4 20	add rsp,20	
00007FF6F3C7146A	5D	pop rbp	
00007FF6F3C7146B	C3	ret	
00007FF6F3C7146C	55	push rbp	

Debugger check patched to xor al, al (Address: 0x00007FF6F3C71463)

00007FF6F3C7145F	FFD0	call rax	
00007FF6F3C71461	85C0	test eax,eax	
00007FF6F3C71463	30C0	xor al,al	
00007FF6F3C71465	90	nop	
00007FF6F3C71466	48:83C4 20	add rsp,20	
00007FF6F3C7146A	5D	pop rbp	

Password check before patch (Address: 0x00007FF6F3C7160A)

00007FF6F3C71600	48:89C1	mov rcx, rax	
00007FF6F3C71603	E9 28180000	call crackme.7FF6F3C72E30	
00007FF6F3C71608	84C0	test al, al	
00007FF6F3C7160A	74 20	jz crackme.7FF6F3C7162C	
00007FF6F3C7160C	48:8D05 432A0000	lea rax, dword ptr ds:[7FF6F3C74056]	00007FF6F3C74056: "good boy"
00007FF6F3C71613	48:89C2	mov rdx, rax	

Password check patched with NOPs (Address: 0x00007FF6F3C7160A)

00007FF6F3C71600	48:89C1	mov rcx, rax	
00007FF6F3C71603	E9 28180000	call crackme.7FF6F3C72E30	
00007FF6F3C71608	90	nop	
00007FF6F3C71609	90	nop	
00007FF6F3C7160A	90	nop	
00007FF6F3C7160B	90	nop	
00007FF6F3C7160C	48:8D05 432A0000	lea rax, dword ptr ds:[7FF6F3C74056]	00007FF6F3C74056
00007FF6F3C71613	48:89C2	mov rdx, rax	

Patched executable output (Address: N/A)

```
PS C:\Users\alira\OneDrive\Desktop\cracktest\test> .\patched.exe
no debugger found
enter a password
asdads
good boy
```

Results

The patched executable successfully bypasses both the debugger presence check and the password validation routine, allowing access to the success message regardless of input. This demonstrates that both protections were effectively neutralized through binary patching.

Legal & Ethical Considerations

This work was performed exclusively for educational purposes on a publicly available crackme challenge designed for reverse engineering practice. The patched binary will not be redistributed. All modifications and analysis respect the platform’s guidelines.

© 2025 Ali Rashchi. All rights reserved.

Contact: alirashchi7@gmail.com